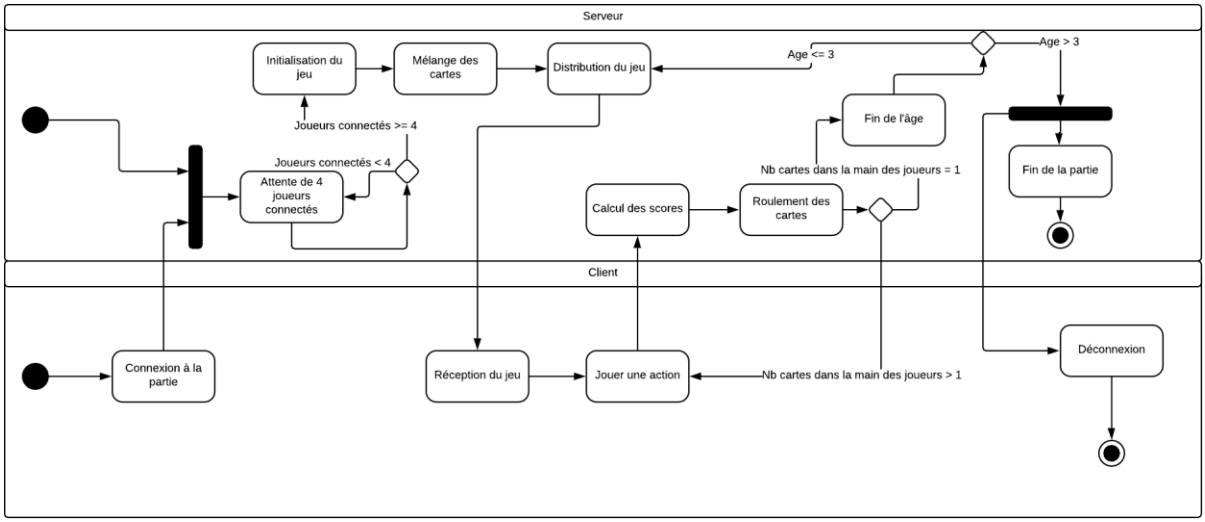
COO
Conception liée au Projet de Développement

GROUPE :

Pierre Saunders
Thomas Gauci
Benoît Montorsi
Rémi Felin
Yannick Cardini

1. Point de vue général de l'architecture

Représentation générale :



Le serveur attend la connexion des joueurs, lorsque 4 joueurs se connectent la partie démarre.
Le client se connecte à une partie et attend que la partie se lance.
Le serveur créer le jeu avec son initialisation et son mélange des cartes.
Puis il distribue le jeu aux joueurs.
Le client réceptionne son jeu et joue une action.
Les joueurs jouent une action, le serveur calcule le score puis ils échangent leurs cartes avec le joueur à gauche ou à droite en fonction de l'âge, jusqu'au moment ou tous les joueurs ont seulement une seule carte en main.
Si il ne reste qu'une carte à chaque joueur alors c'est la fin de l'âge en cours.
Si l'âge n'est pas plus grand que 3 alors on refait les actions précédentes sinon c'est la fin de la partie et le client se déconnecte.

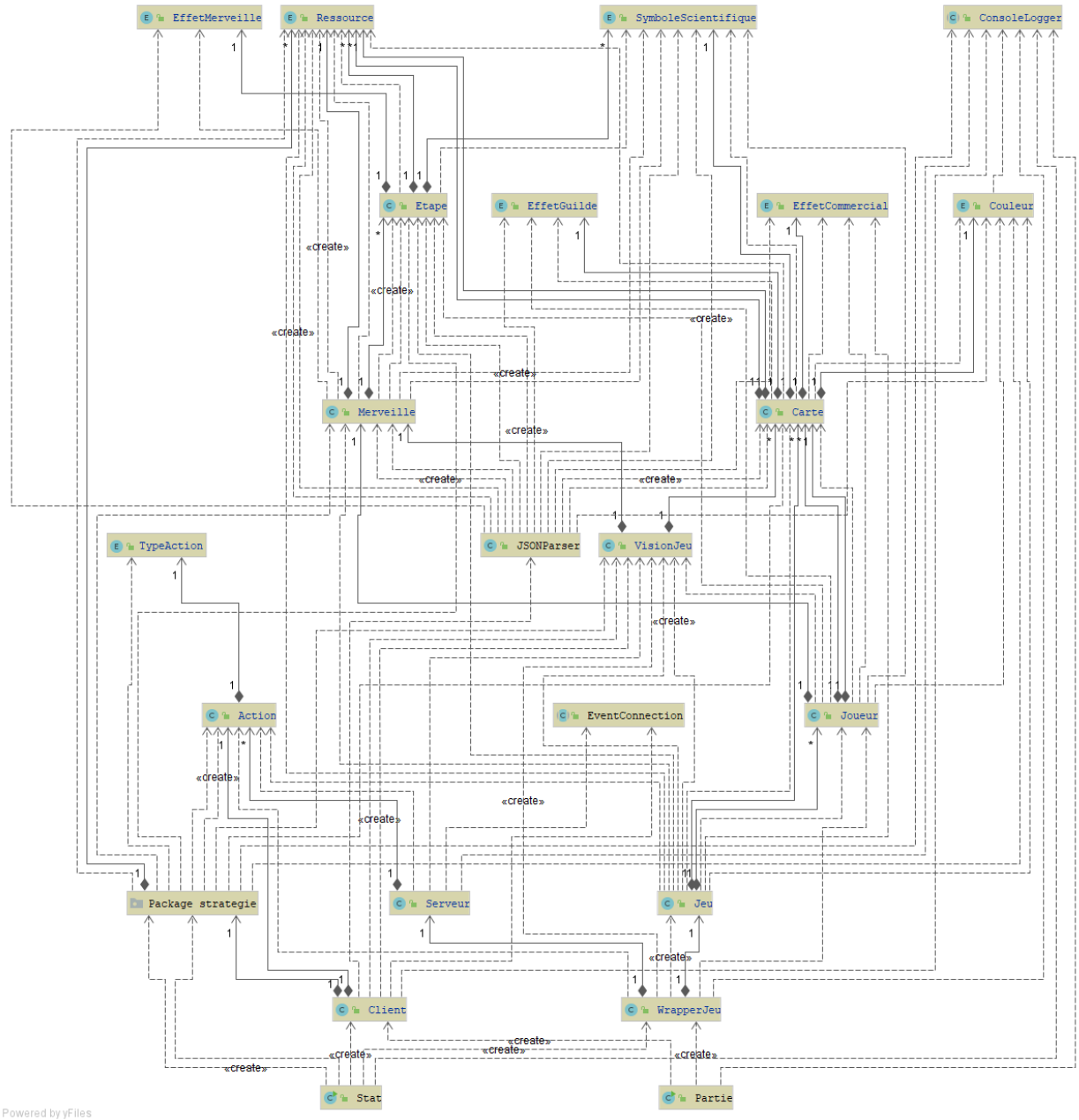
2. Client

• *Analyse des besoins (exigences) : Cas d'utilisation*
    • Acteurs (le client est un acteur du serveur et vice versa)
    • Diagrammes de Cas d'utilisation
    • Scénarios (sous forme textuelle, un par use case)

• *Conception logicielle*
    • Point de vue statique : Diagrammes de Classes lisible en format A4 (une découpe «logique » avec répétition de certaine classe).

Point de vue Dynamique

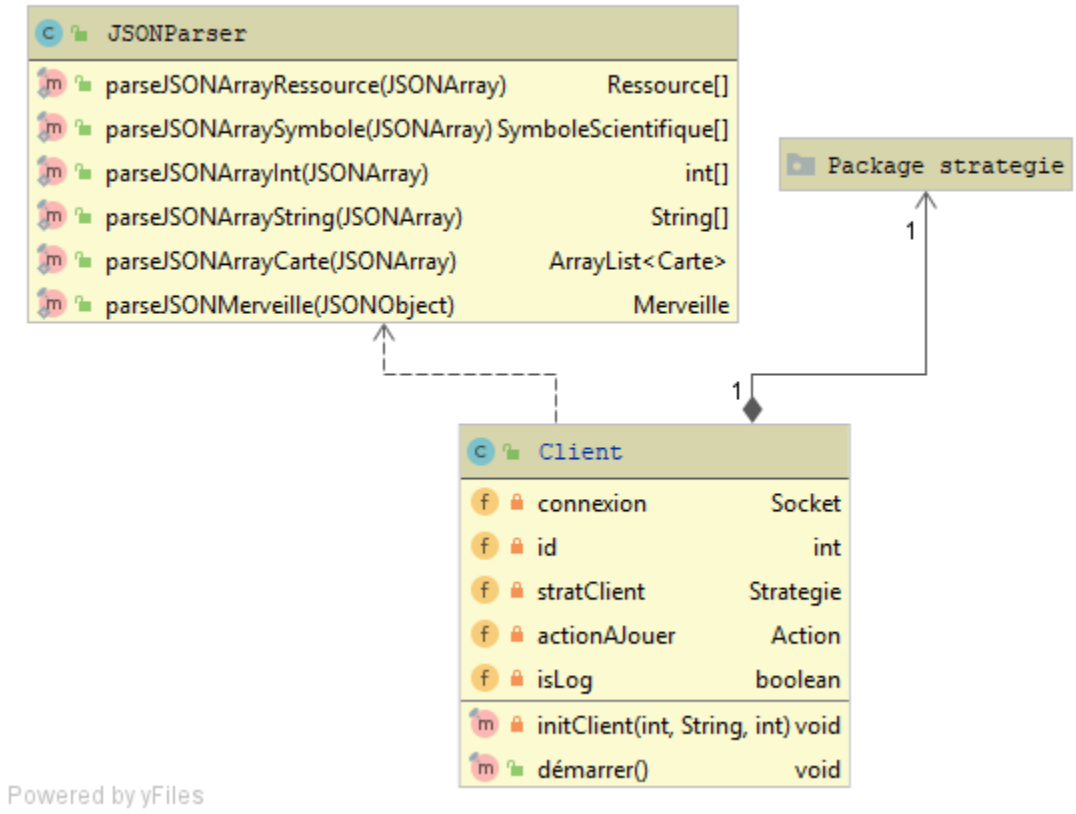
Diagramme de classe général:



Un client ne peut avoir qu'une seule stratégie, et ne peut faire qu'un seul coup (une seule Action) dans un tour. Le serveur n'est lié qu'à un seul jeu, et vice versa. Le jeu contient plusieurs joueurs, et plusieurs decks composés de plusieurs cartes. Celles-ci peuvent

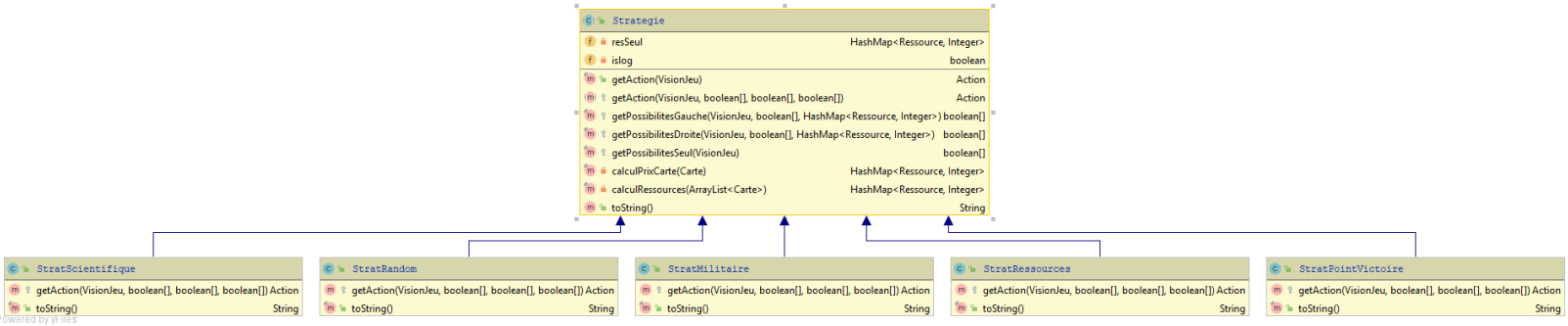
contenir 1 ou plusieurs bonus, et possèdent un coût en pièce et/ou en ressources. Chaque merveille possède un certain nombre d'étapes (allant de 2 à 4 étapes).

Diagramme du module client:



Un client (ou un bot, dans les faits) doit pouvoir se connecter via une socket. Chaque client possède un id spécifique qui permet leur identification par le serveur. Chacun a une action spécifique à faire à chaque tour, liée à une stratégie particulière.
La distinction en plusieurs stratégies (présentes dans le package indiqué, et sur le diagramme suivant) permet théoriquement une action différente pour chaque client. Un JSONParser est nécessaire pour pouvoir traiter l'information transmise avec socket io.

Diagramme du module “strategie” :

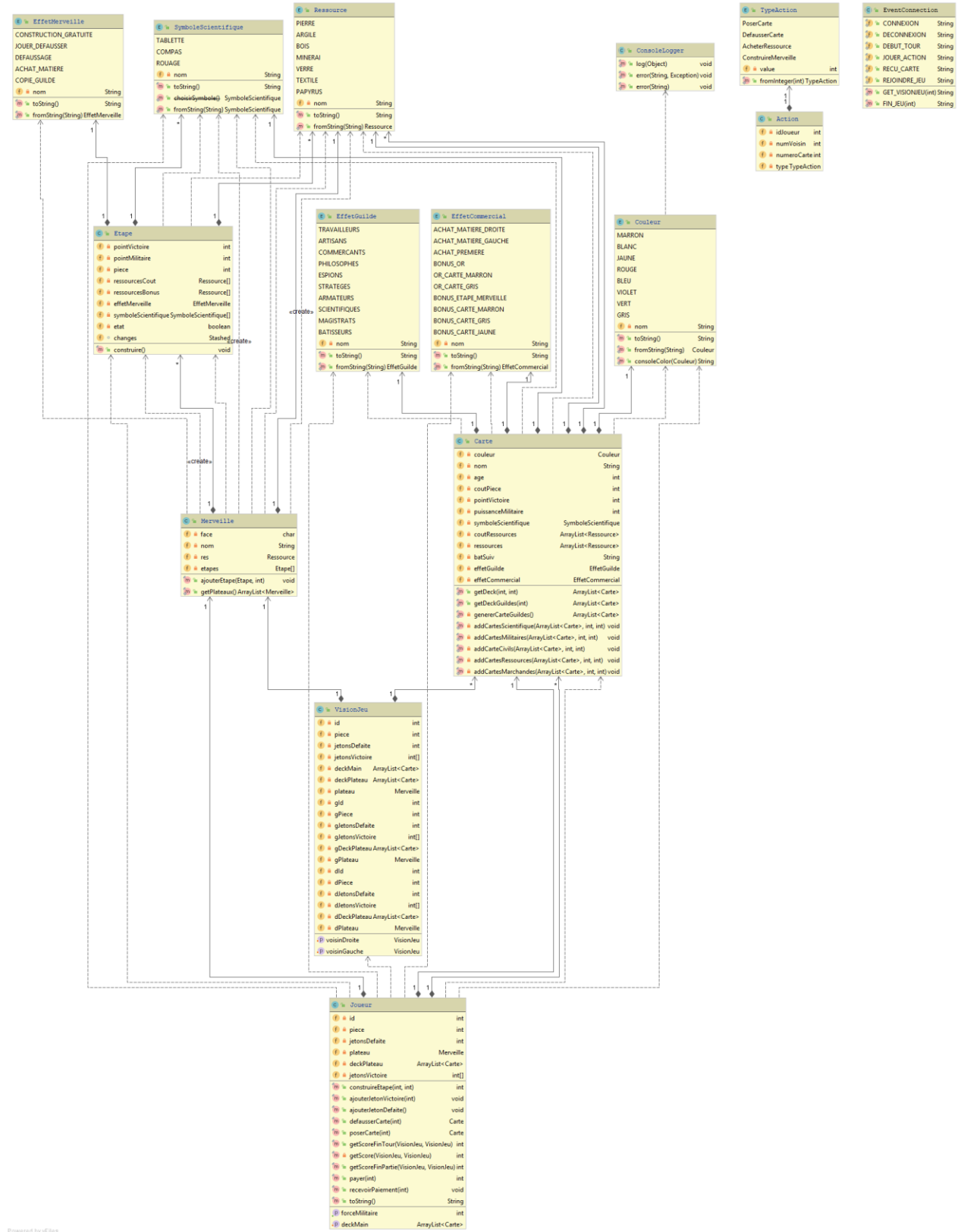


Les clients (les bots) possèdent une stratégie décidée aléatoirement.
5 stratégies sont disponibles:

- une basée sur l’accumulation de cartes victoire (StratPointVictoire)
- une jouant des cartes aléatoirement (StartRandom)
- une se basant sur l’accumulation de points militaires (StratMilitaire)
- une se basant sur l’accumulation de ressources (StratRessources)
- une basée sur l’accumulation de symboles scientifiques (StratScientifique).

Afin de faire une Action correspondant à une certaine stratégie, le Client va d’abord récupérer toutes les possibilités d’Action (“quelles Cartes peut-on jouer avec notre jeu seul?”, “quelles Cartes peut-on acheter chez les voisins afin de jouer une certaine Carte de notre deck?”). Une fois celles-ci récupérées, la stratégie décide quelle serait la carte la plus appropriée à la situation du Client, et renvoie l’Action choisie au Serveur.

Diagramme du module commun:



Dans le jeu, il y a différentes ressources: pierre, argile, bois…

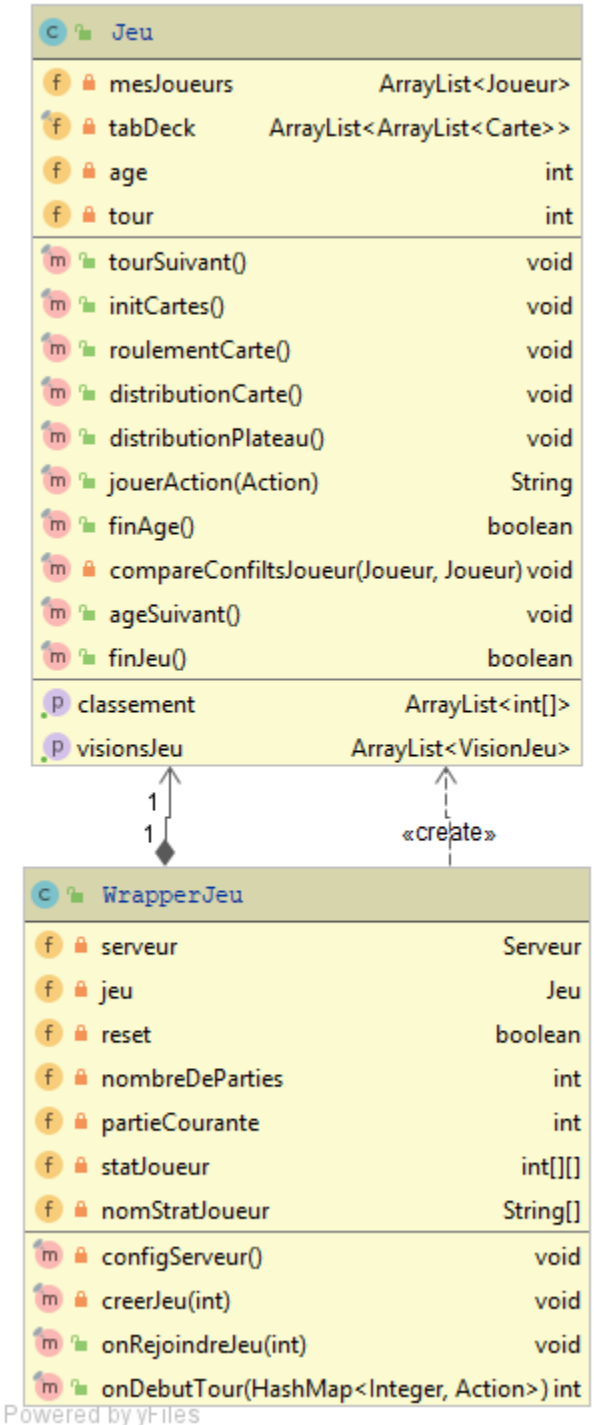
Une carte peut posséder différentes caractéristiques: un coût en ressources ou en pièces à l'utilisation, un nom unique et une Couleur. Lorsqu'on la joue, elle apporte un bonus variable en pièces, en puissance militaire, en ressources, ou en lauriers, ou un autre Effet Commercial.

Selon le déroulement du jeu, une carte peut permettre la construction d'un certain "bâtiment", octroyant certains avantages, selon certaines conditions spécifiques.

Enfin, plus la partie avance (avec le système d'âges), plus les cartes peuvent apporter des effets de guilde.

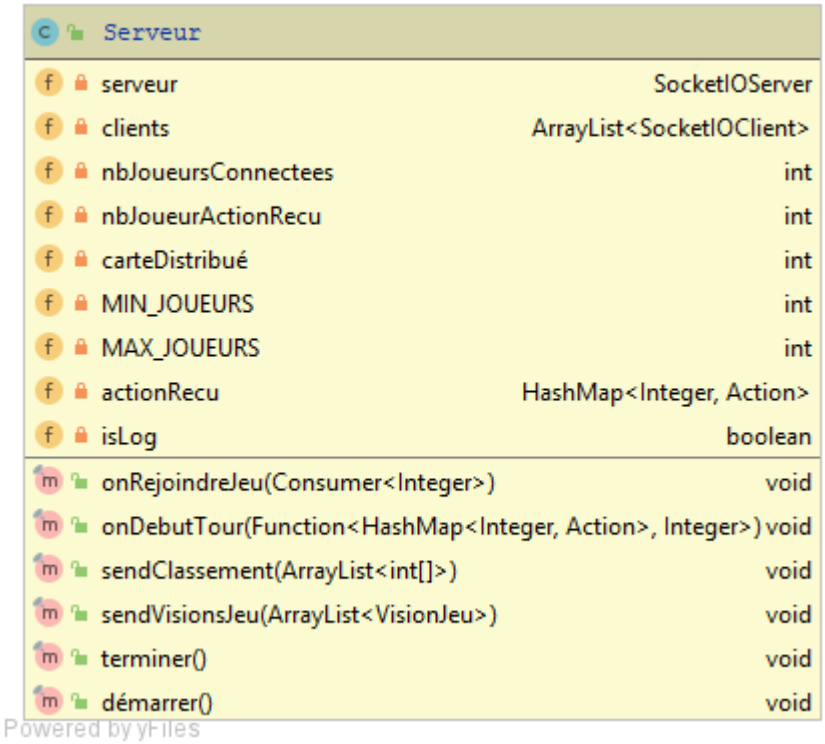
Un Joueur peut posséder une ou plusieurs cartes. Il a un id spécifique, permettant son identification, et possède son propre plateau (Merveille) et un certain nombre de pièces. Après un conflit militaire, il se voit octroyé d'un ou de plusieurs jetons de victoire ou de défaite.
Durant un tour, il peut poser (jouer) une carte, ou en défausser une. Il peut construire une Étape d'une Merveille, si les conditions à sa construction sont remplies. Chaque Merveille possède un effet spécial.
Enfin, chaque Joueur possède une Vision de jeu, sur ses voisins.

Diagramme du module moteur:



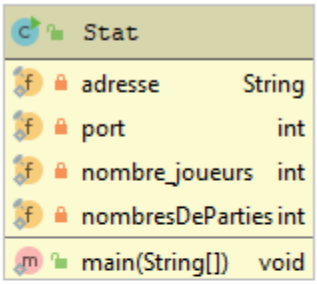
Un Jeu contient plusieurs joueurs. Il sait combien de cartes il faut distribuer.
D'une manière générale, il fait fonctionner le jeu, c'est à dire: initialiser les cartes, les distribuer, faire jouer un joueur, et comparer les scores afin d'établir le classement général.
Le Jeu doit aussi avoir accès aux joueurs eux-mêmes. En fonction du Type de l'Action, le Jeu effectuera une certaine Action. A la fin d'un âge, le Jeu se charge des conflits militaires entre les Joueurs.
Le wrapperJeu permet de lancer le jeu avec le Serveur.

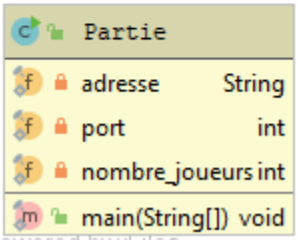
Diagramme du module serveur:



Notre serveur doit interagir avec les clients: contrôler le nombre de joueurs connectés afin de pouvoir commencer une partie, signaler le début et la fin des tours, des âges et du jeu avec toutes les opérations qui s'en suivent.

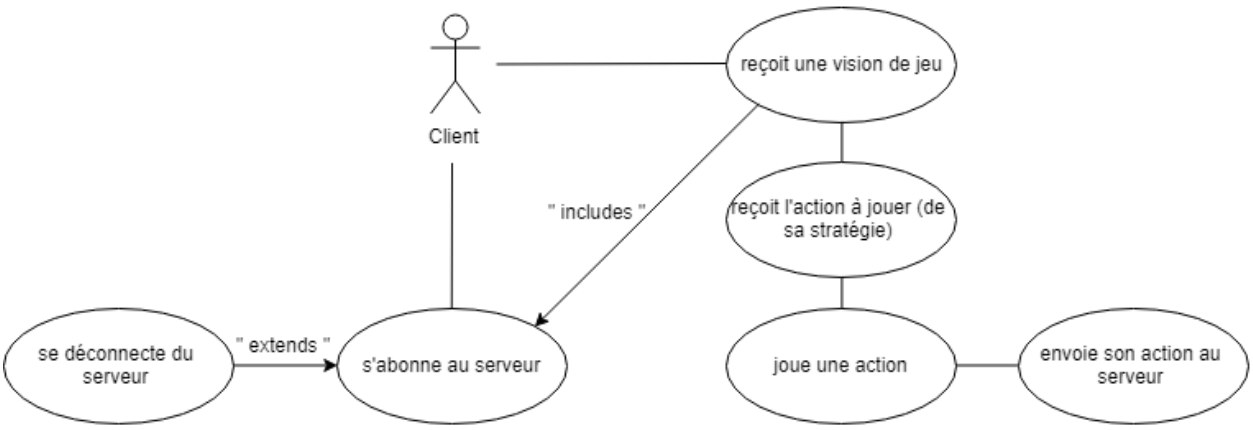
Diagramme du package lanceur:



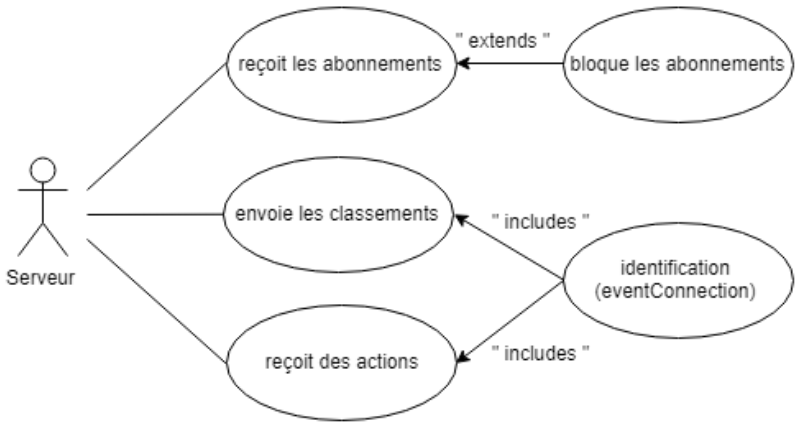


Le Lanceur doit lancer le serveur afin que les clients puissent s'y connecter. Il crée également nos clients (les bots), selon un nombre de joueurs qui lui est indiqué.
Pour faire tout cela, il doit posséder une certaine adresse pour lancer le serveur à cette adresse, et un port spécifique.
On peut lancer une seule partie avec Partie., tandis que Stat nous permet de lancer autant de parties que l'on veut (ici 500).

Use case :



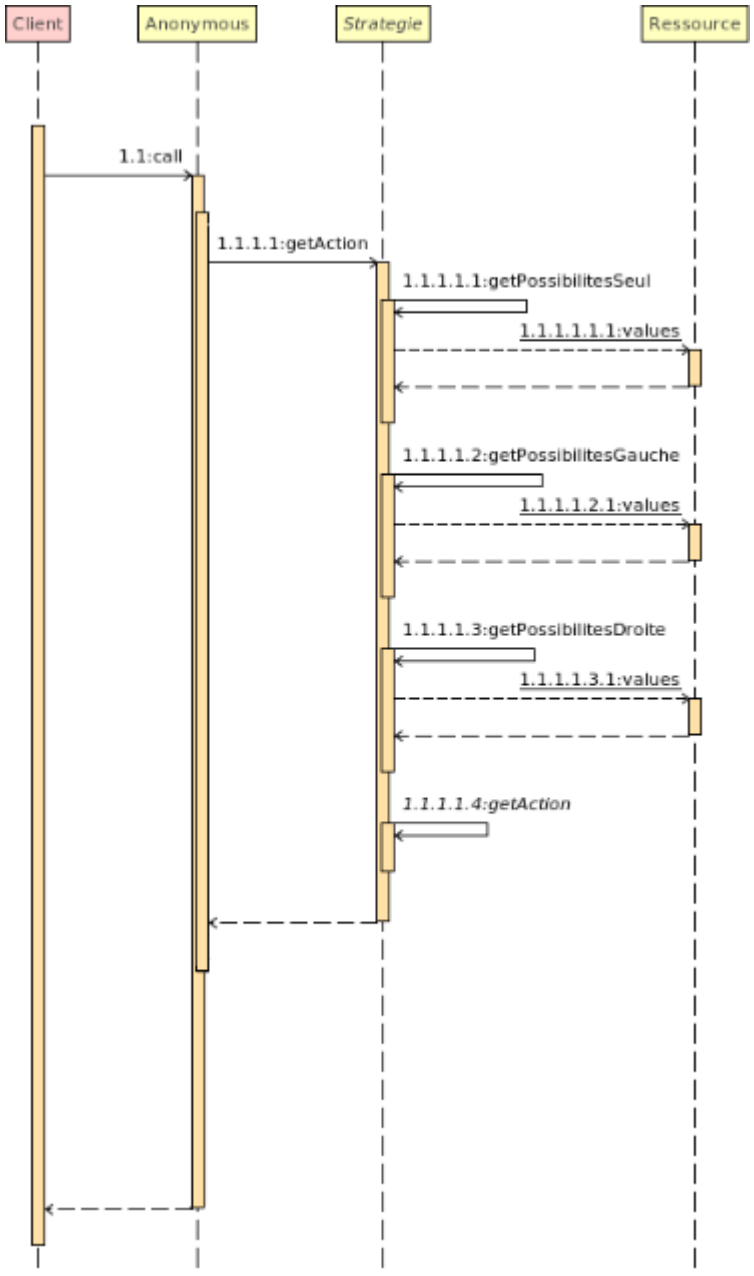
Notre client (s'il est connecté au serveur) reçois une vision de jeu (cette vision est envoyé à sa stratégie), Par la suite, notre client reçoit l'action à jouer de la part de sa stratégie, il joue donc la dite action et l'envoie au serveur (et ainsi de suite). Notre client peut également se déconnecter du serveur (s'il est déjà connecté)



Le serveur reçoit les abonnements, il peut également se permettre de bloquer les abonnements reçus si le nombre de joueurs max venait à être dépassé. Lors de la partie, le serveur reçoit les action des joueurs. Enfin, le serveur envoie les classements à la fin de chaques âges ainsi qu'à la fin du jeu (sous réserve qu'il soit authentifier).

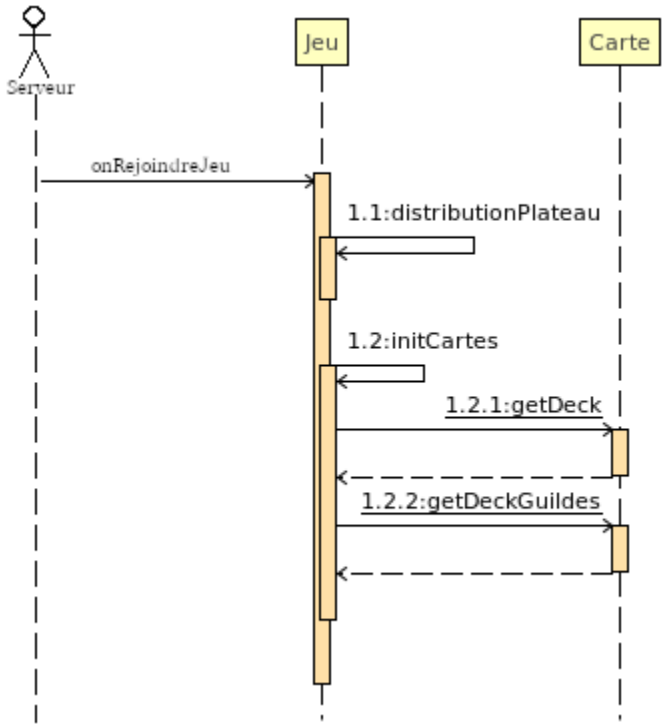
Diagramme de séquence:

Client:



La client (en rose) correspond à l'élément déclencheur, c'est lui qui appelle la classe Stratégie pour savoir quelle action réaliser.
Stratégie choisit une stratégie à adopter en fonction des Ressources de ses voisins et renvoie l'action adopter au Client.

Jeu:



Une fois tous les joueurs connectés, la fonction de callback "*onRejoindreJeu*" est appelée et instancie un nouveau jeu.
On commence par instancier les plateaux avant d'instancier un deck de carte, puis un deck de guilde.

4. Conclusion

Points forts :

- Structuration du projet
- Client/Serveur sur un seul buffer

Points faibles :

- Problème de communication avec le client